注：本次调试步骤为以commit 417117eac3和7.29日最新的commit作为对比参照

复现`test_wint8()`单测

首先，复现`test_wint8()`这个单测错误的命令为

```
# current's dir is PaddleNLP/
python -m pytest -s -v
tests/llm/test_predictor.py::BlockAttnPredictorTest_0___internal_testing___tiny_
fused_llama_inference5_2::test_wint8 > test_check.log 2>&1

# export test_wint8() model
python export_model.py --model_name_or_path /root/autodl-
tmp/models/models/__internal_testing__/tiny-fused-llama-inference5.2 --
output_path /root/autodl-tmp/models/exported_model/tiny-fused-llama-inference5.2
 --dtype float16 --inference_model --block_attn 1 --quant_type 1
```

由宏观到微观的精度对齐

Step 1 逐层打印并检查

最开始需要从宏观的角度来查看每一层的output tensor的值，可以再`predictor.run()`之前加入`hook_func`

```
def hook_func(op_type: str, tensor_name: str, tensor: paddle.Tensor):
    # ... 在这里可以求和、求均值、求方差或者直接输出全部数据等等
    print(op_type, tensor_name, tensor)
    # ...

predictor.register_output_hook(hook_func)
```

这里，是需要给`StaticBlockInferencePredictor`这个类中`_create_predictor`这个函数最后一行加代码如下

```
def hook_func(op_type: str, tensor_name: str, tensor: paddle.Tensor):
    print(op_type, tensor_name, tensor)
self.predictor.register_output_hook(hook_func)
```

这里可以找出第一次出错的tensor，由于我最开始认定block_attn有精度问题，因此，主要检查这个算子的输出，最终在第10个block_attention发现出错。

Step 2 每次block_attention算子中打印中间变量

通过VLOG和VLOGMatrix函数进行打印，主要打印的位置及代码如下

```
/*
*
path:/home/Paddle/paddle/phi/kernels/fusion/gpu/block_multi_head_attention_kerne
l.cu
* 刚进入DispatchWithDtype函数时
*/
```

```
    DenseTensor* value_cache_out) {
  phi::DenseTensor qkv_buf;
  phi::DenseTensor fmha_buf;
  VLOGMatrix(qkv.data<T>(), 10, "***qkv pos1***", 10);
  VLOG(1) << "fmha_out " << fmha_out->dims();
  if (out_scale <= 0) {
    dev_ctx.template Alloc<T>(fmha_out);
    fmha_buf = *fmha_out;
  } else {
    fmha_buf.Resize(fmha_out->dims());
    dev_ctx.template Alloc<T>(&fmha_buf);
    dev_ctx.template Alloc<int8_t>(fmha_out);
  }
  VLOGMatrix(fmha_buf.data<T>(), 10, "***fmha_buf pos1***", 10);

  // InitValue(dev_ctx, fmha_buf.data<T>(), fmha_buf.numel(), static_cast<T>
(0.));
  VLOGMatrix(fmha_buf.data<T>(), 10, "***fmha_buf pos11***", 10);
  const auto& input_dims = qkv.dims();

/*
* 退出DispatchWithDtype函数之前
*/
if ((*qkv_out).initialized()) {
    VLOGMatrix((*qkv_out).data<T>(), 10, "qkv_out_buf final", 10);
} else {
    LOG(INFO) << "qkv_out is nullptr";
}

if ((*fmha_out).initialized()) {
    VLOGMatrix((*fmha_out).data<T>(), 10, "fmha_out_buf  final", 10);
} else {
    LOG(INFO) << "fmha_out is nullptr";
}


if ((*key_cache_out).initialized()) {
    VLOGMatrix((*key_cache_out).data<T>(), 10, "key_cache_out  final", 10);
} else {
    LOG(INFO) << "key_cache_out is nullptr";
}

if ((*value_cache_out).initialized()) {
    VLOGMatrix((*value_cache_out).data<T>(), 10, "value_cache_out  final", 10);
} else {
    VLOG(3) << "value_cache_out is nullptr";
}
```

Step 3 根据打印log分析出错位置

(需要根据导出的pdmodel来追溯出现错误的起因)

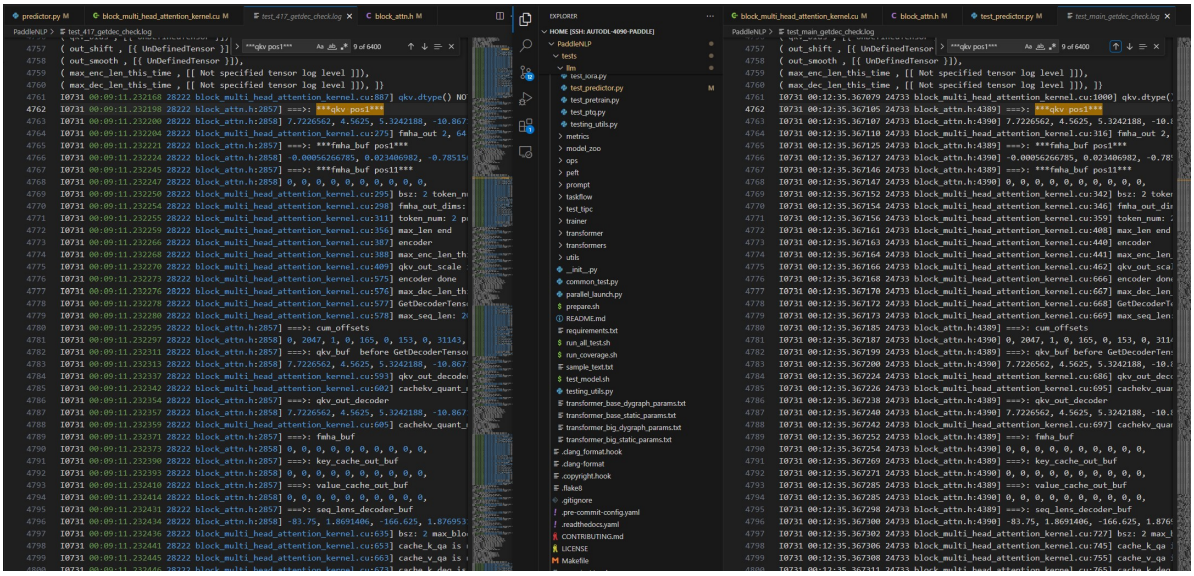
在打印了一些变量的值后可以发现，在第10轮计算block_attn的时候，输入的qkv就已经开始有误差了。

变量打印的位置如下：

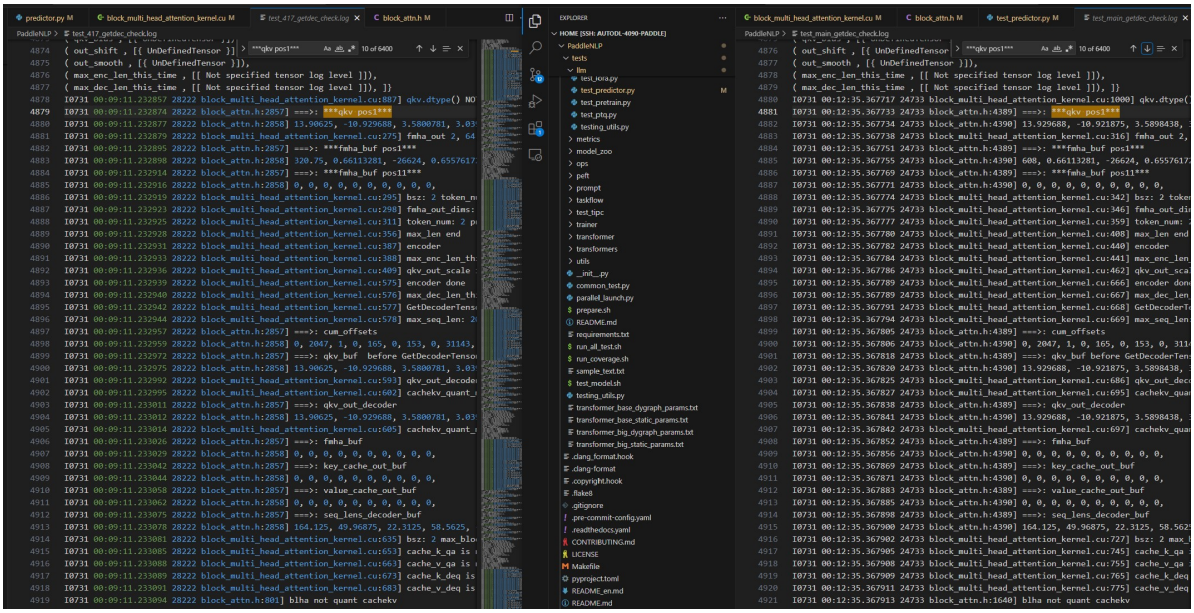
```
    const int quant_round_type,
    const float quant_max_bound,
    const float quant_min_bound,
    const float out_scale,
    const std::string& compute_dtype,
    DenseTensor* fmha_out,
    DenseTensor* qkv_out,
    DenseTensor* key_cache_out,
    DenseTensor* value_cache_out) {
  phi::DenseTensor qkv_buf;
  phi::DenseTensor fmha_buf;
  VLOGMatrix(qkv.data<T>(), 10, "***qkv pos1***", 10);
  VLOG(1) << "fmha_out " << fmha_out->dims();
  if (out_scale <= 0) {
    dev_ctx.template Alloc<T>(fmha_out);
    fmha_buf = *fmha_out;
  } else {
    fmha_buf.Resize(fmha_out->dims());
    dev_ctx.template Alloc<T>(&fmha_buf);
    dev_ctx.template Alloc<int8_t>(fmha_out);
  }
  VLOGMatrix(fmha_buf.data<T>(), 10, "***fmha_buf pos1***", 10);

  InitValue(dev_ctx, fmha_buf.data<T>(), fmha_buf.numel(), static_cast<T>(0.));
  VLOGMatrix(fmha_buf.data<T>(), 10, "***fmha_buf pos11***", 10);
  const auto& input_dims = qkv.dims();
```

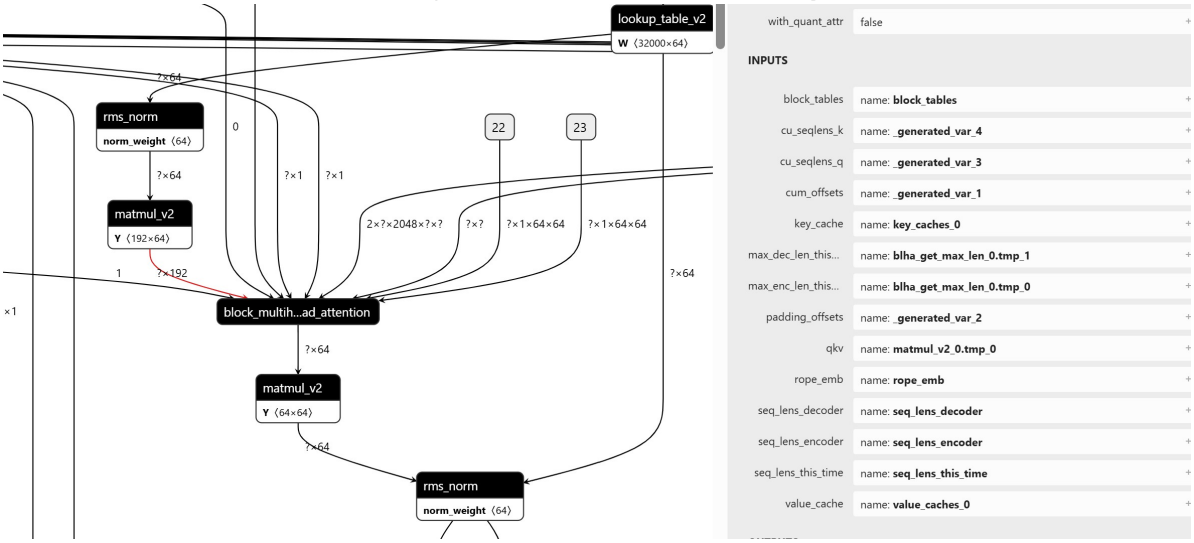
第九轮一些变量的值是这样的，可以发现**所有tensor的值都是一样的**



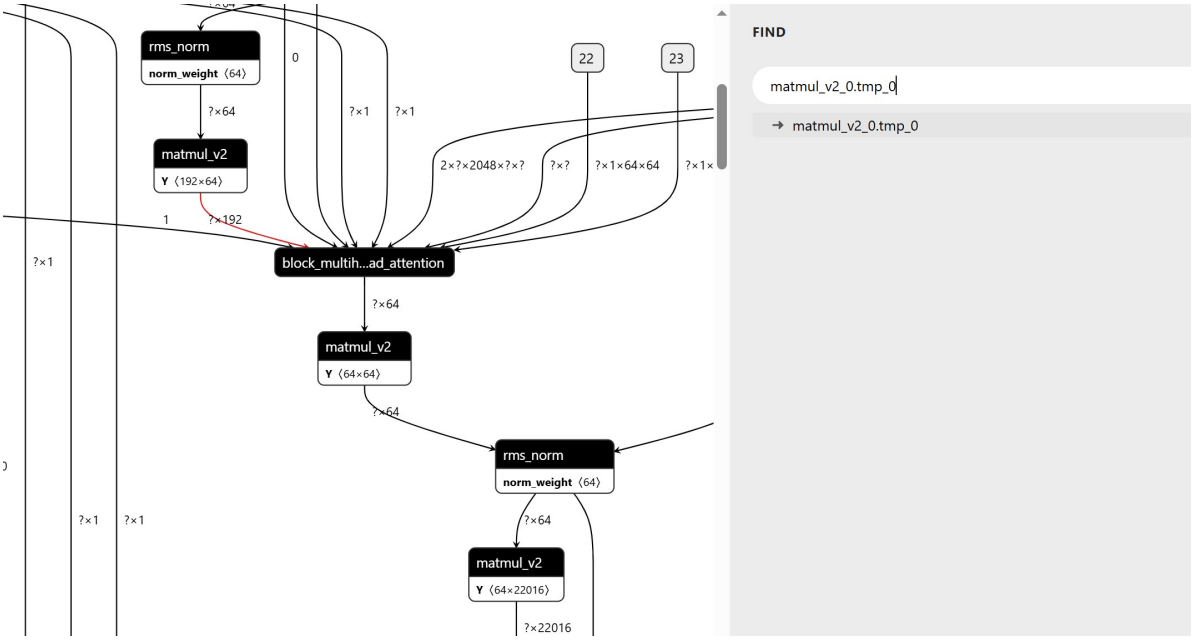
第10轮，可以发现已经**有一些tensor的值出现了微小的精度差异**，追根溯源，发现在**一进来的时候qkv的值就不一样**了。



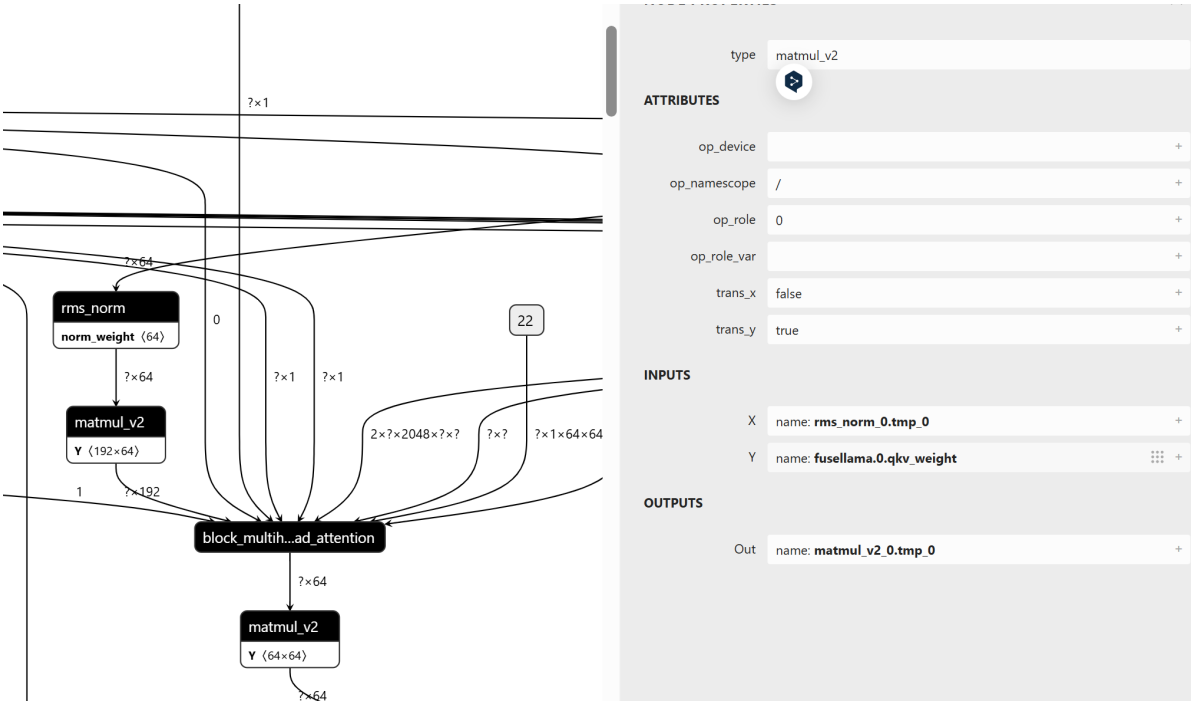
然后去netron中查找，可以发现这个qkv在图中名为**matmul_v2_0.tmp.0**



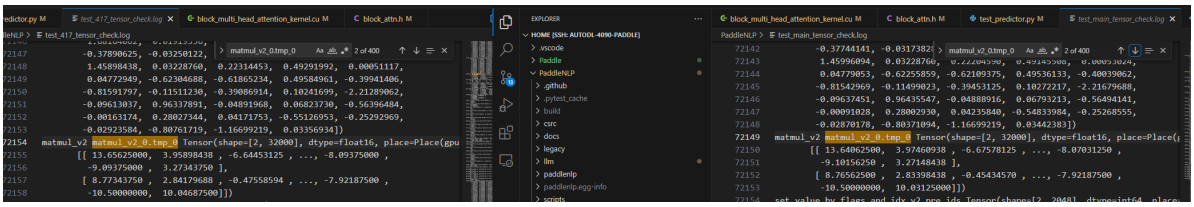
它来自于block_multihead_attention的上一个matmul。



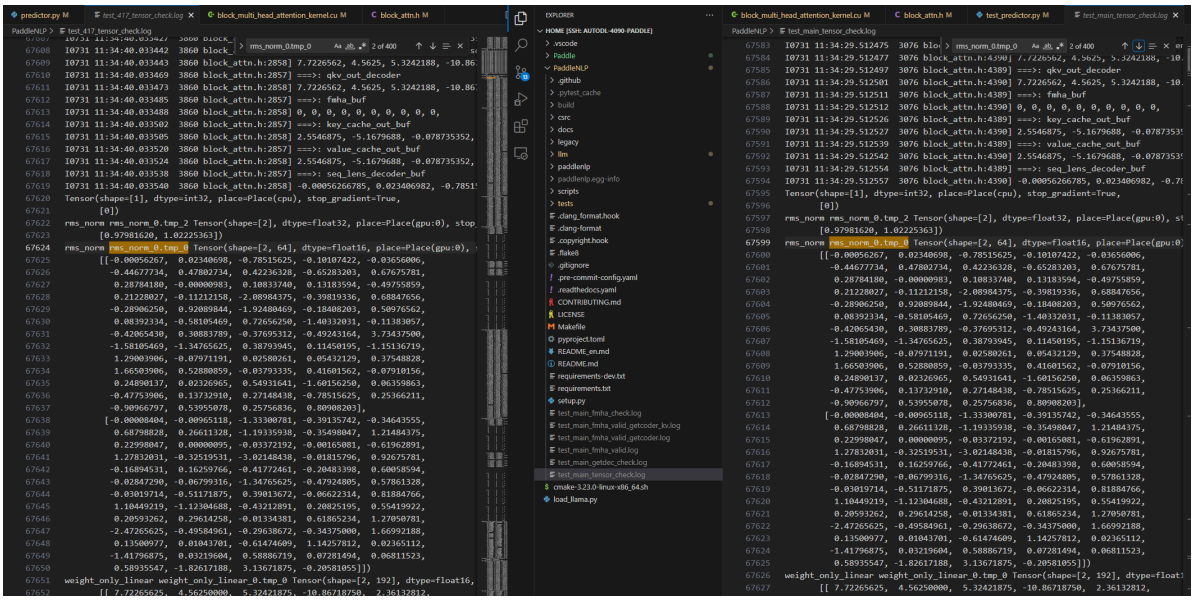
这里weight假设是肯定没有精度差异的。

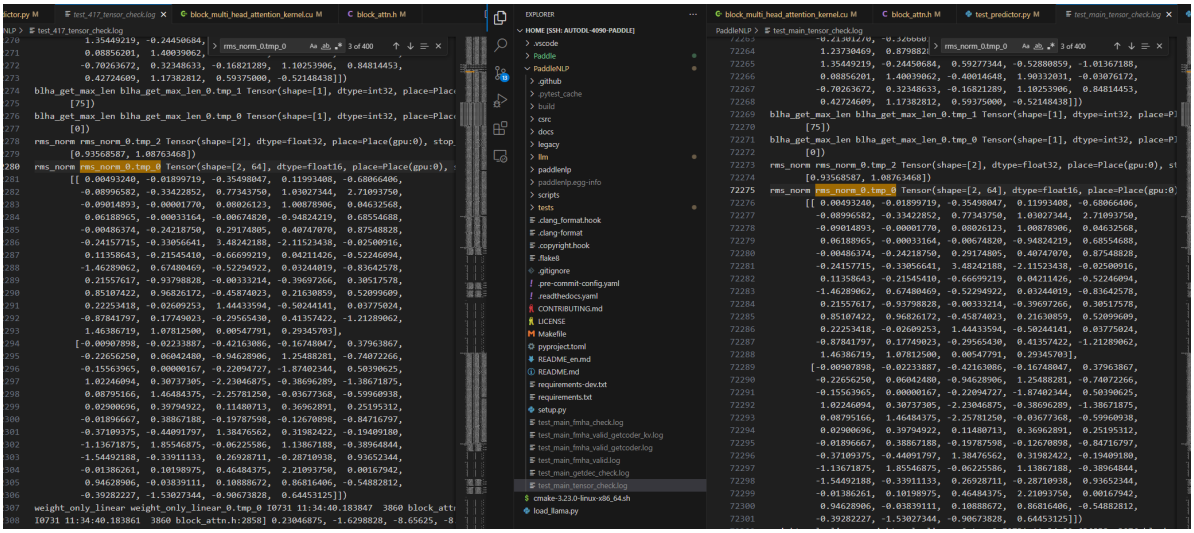


去打印出来的tensor看，发现第二次计算得到的matmul_v2_0.tmp_0精度就不一样了。



因此继续去看到matmul_v2的输入rms_norm_0.tmp_0





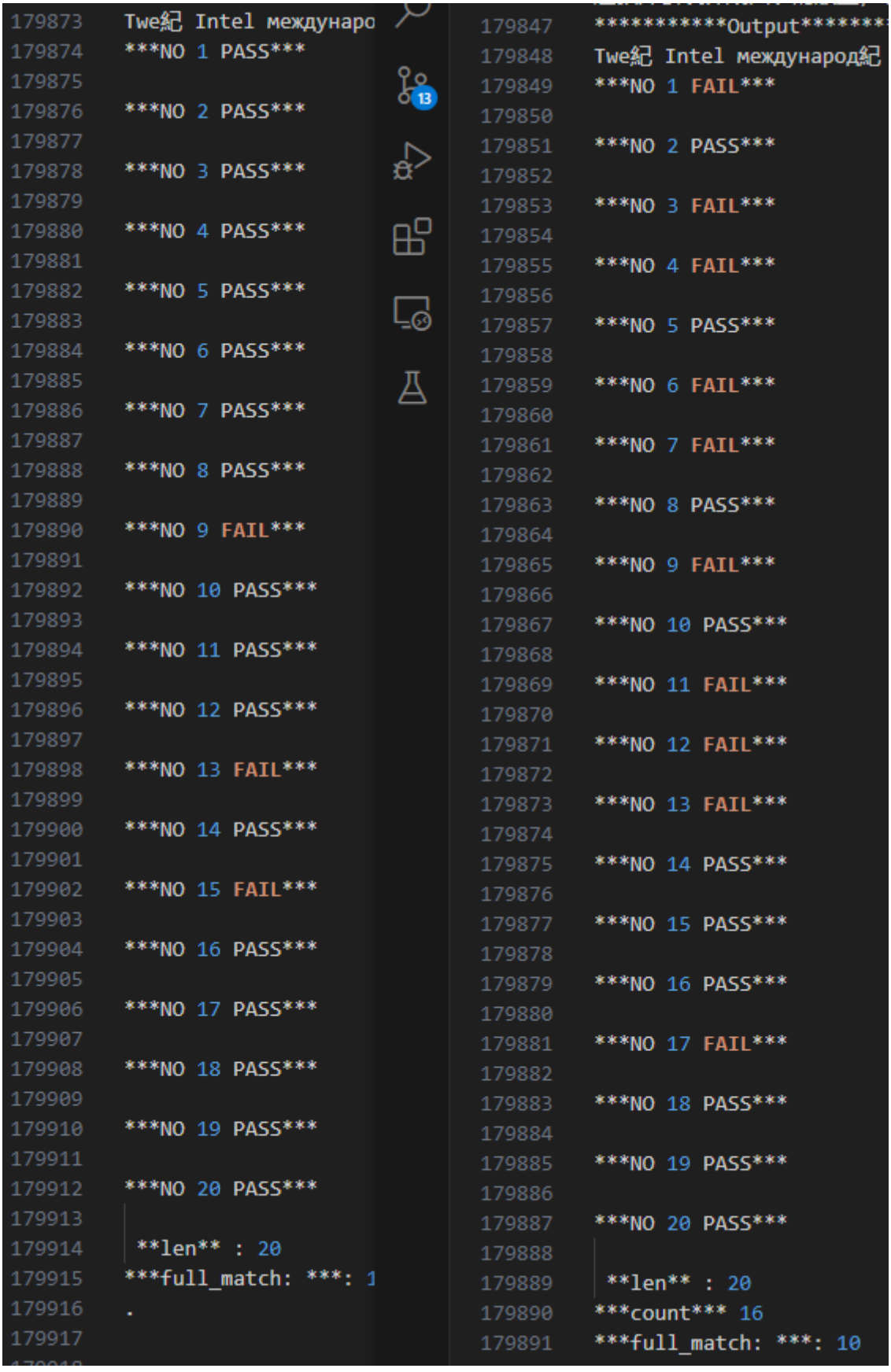
可以发现，在所有数据中的第2和第3个rms_norm_0.tmp_0均无精度问题，（可以通过看行数发现第3个rms_norm_0.tmp_0已经在第二次计算得到的matmul_v2_0.tmp_0之后了）因此，目前推断matmul_v2有精度误差。而根据对比commit id 417...的逻辑，develop分支最新的commit下test_wint8单测的精度问题不在block_attn。

每个case fail和pass的情况

通过对单个不同input逐个分析发现，两个commit之前的对比如下（左边为417...,右边为develop最新commit)

这里batch size为2，也就是说20个test case，每个case跑两个input，实际跑的时候，共20个句子，每个句子有两次作为输入，第1-10次推理，每次两个句子，共跑10次，第11-20次推理，同上。

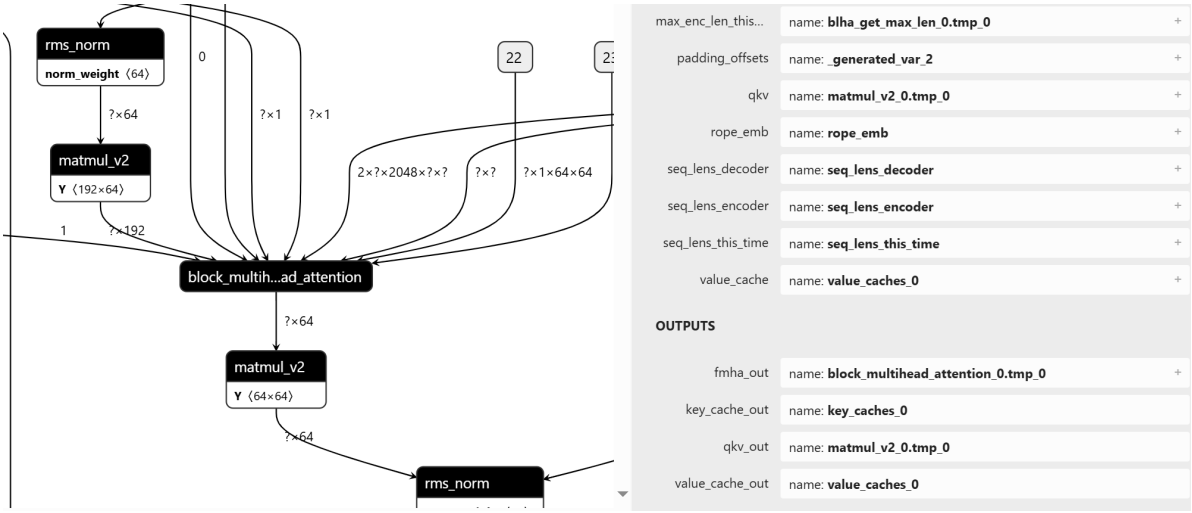
实际，判断的时候，用另一个命令把block_attn设置flase，再把两个命令输出的句子进行对比。



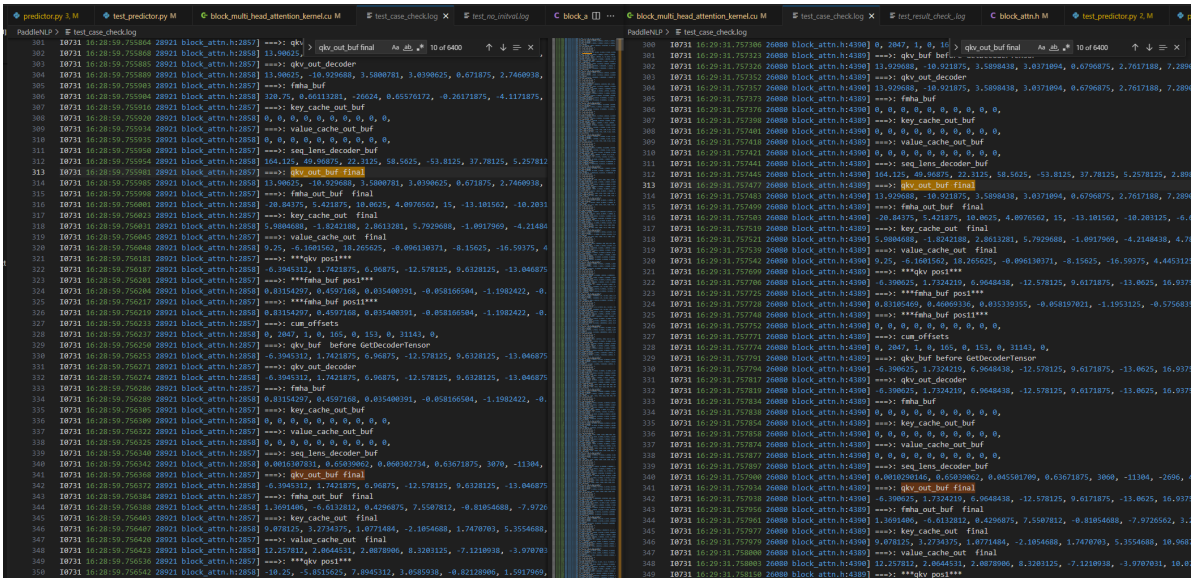
左边过17个，右边过10个，根据之前的分析（用417...的第1个case比对最新commit的第1个case），发现精度的问题从matmul开始对不齐，但是经过后续分析可以发现，即使是用417...的第2个case比对最新commit的第2个case，也就是两个commit都能过的case，在后续也会出现同样的精度无法完全对齐的问题，可是单测都过了，也就是说在两边都能过的case中，我理解最终生成的token_id受这些精度问题的影响不是特别大。

根据上述想法，再打印一些更多的tensor数据出来，来判断上述想法的可能性。

首先，看看block_attn的最终输出在这些case中的区别，根据下图，block_attn的输出有四个，都打印出来。

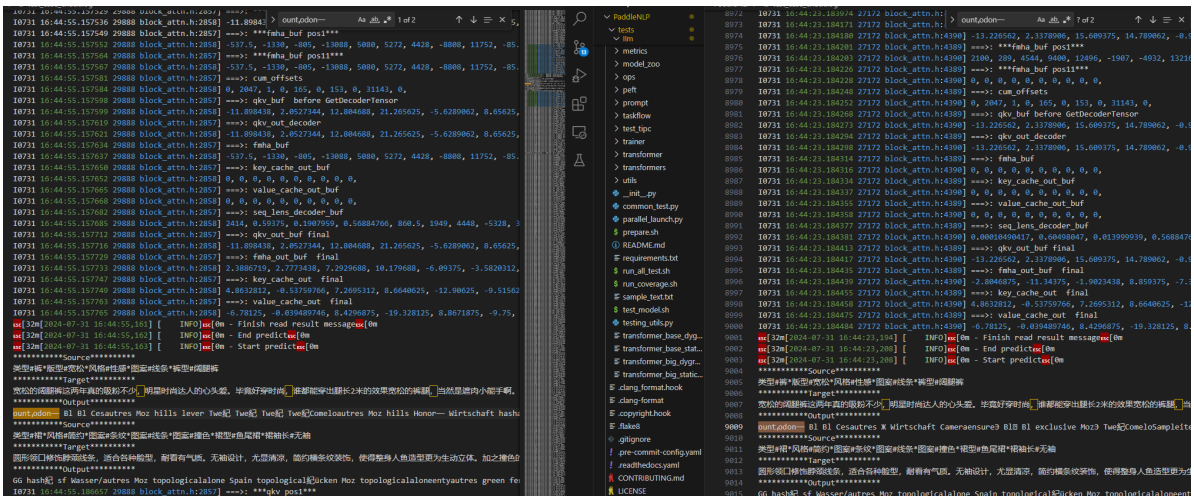


第1个case（417-id，PASS，new-id，FAIL）NO1&NO2



和之前的分析一样，最开始的几个轮次精度正常，从第10轮开始出现精度问题.

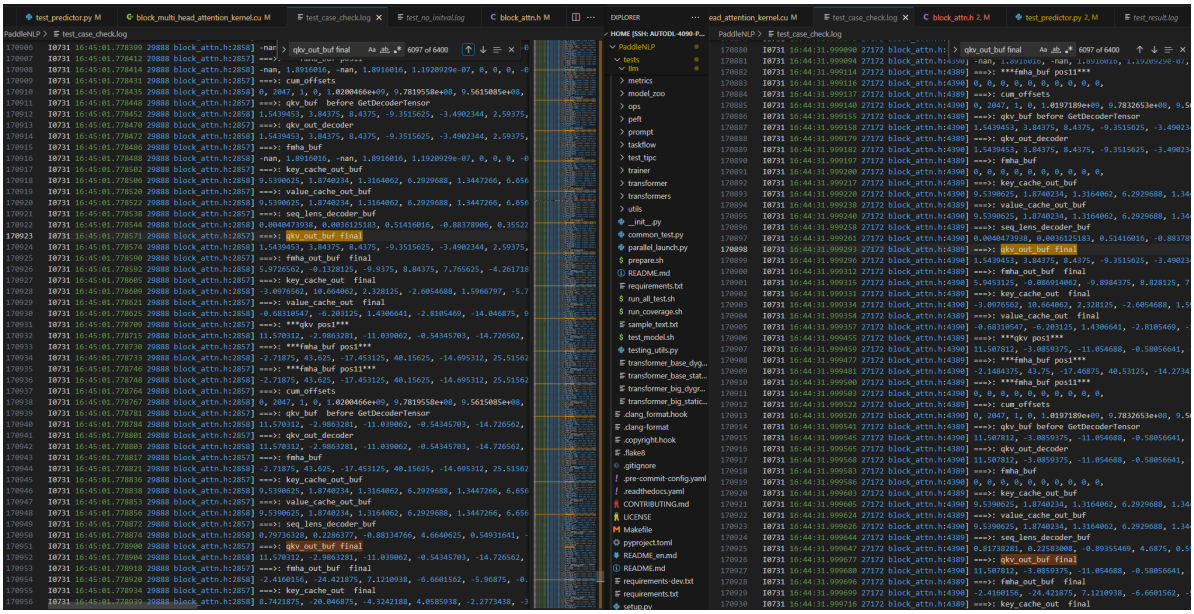
看看该case下的最终输出，最终输出中key_cache和value_cache一样，qkv_out个别元素有较大误差，绝对误差在[0, 5]之间，fmha_out误差较大在[0,15]之间



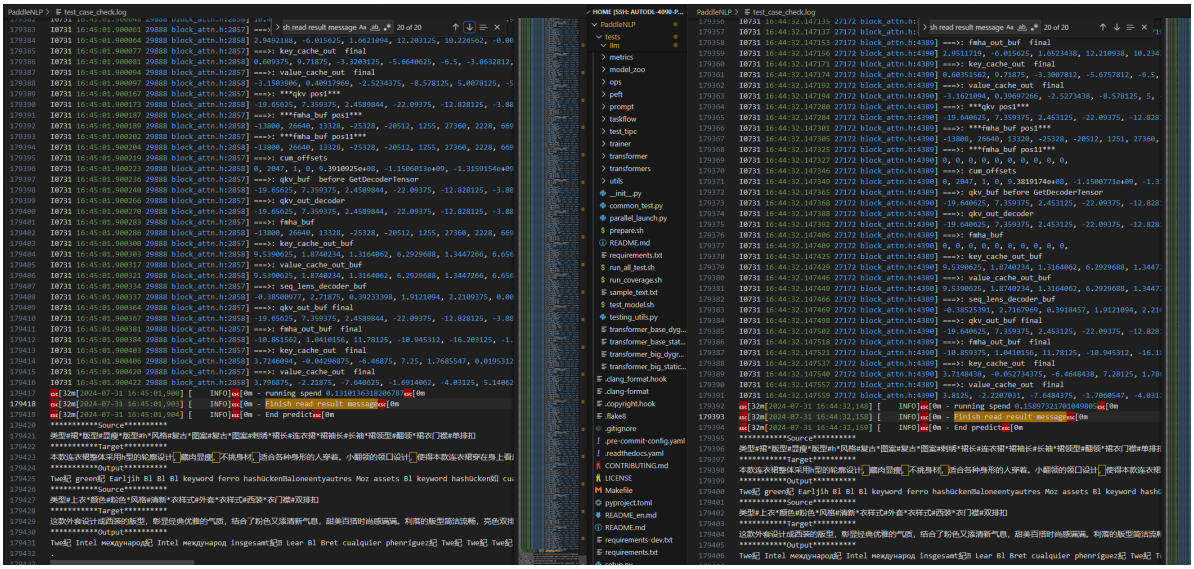
这个case实际输出了两个output，第一个input pass，第二个input fail

第2个case（417-id，PASS，new-id，PASS）ALL PASS NO19&NO20

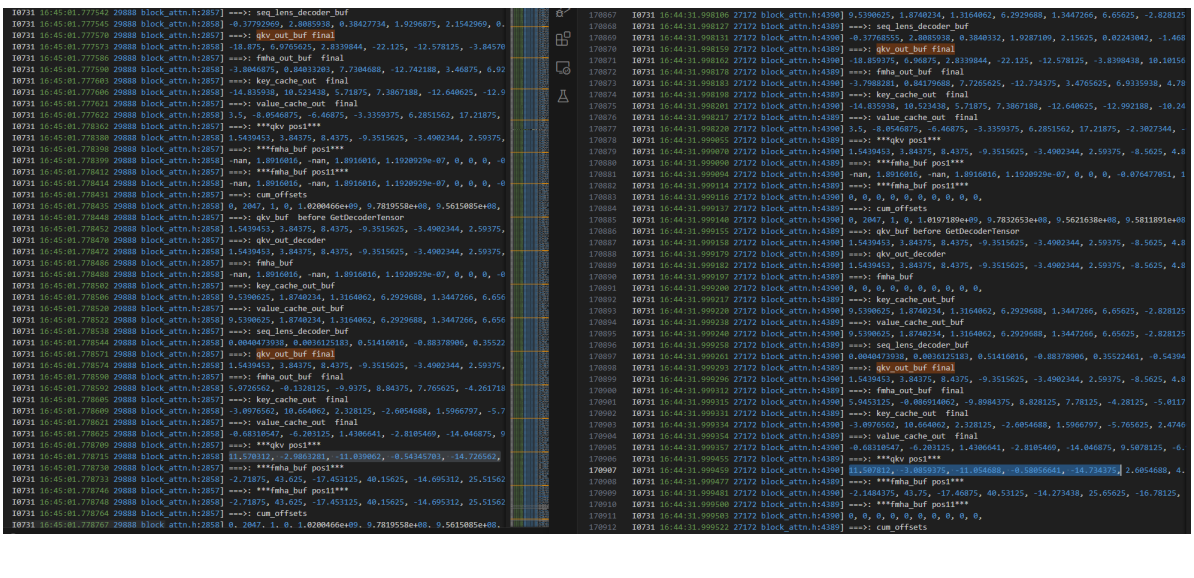
情况同第一个，虽然都该case两个commit都pass了，但是也存在精度误差。从第17轮开始出现精度问题



最终的输出，有误差，但不大，两个input都pass

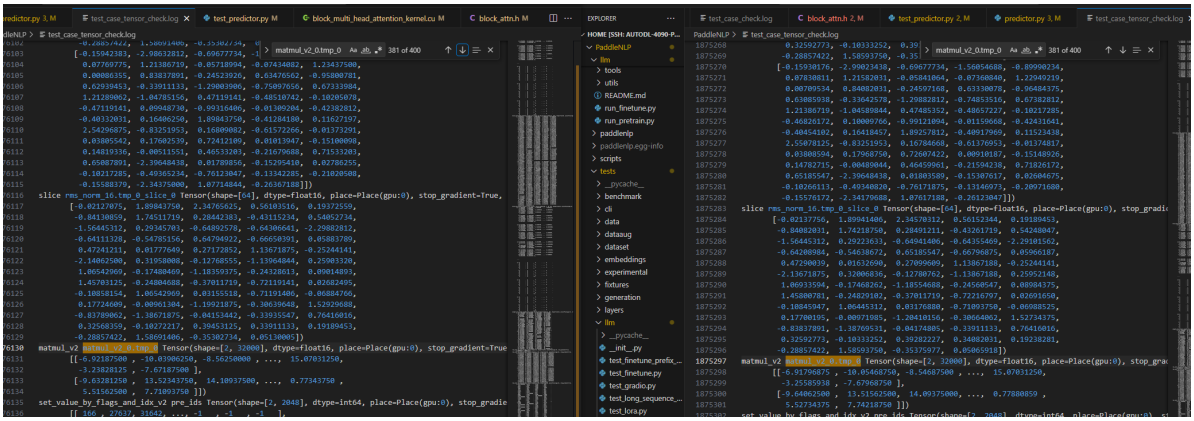


情况一样的是，这里的误差，也来自于该次层input的qkv

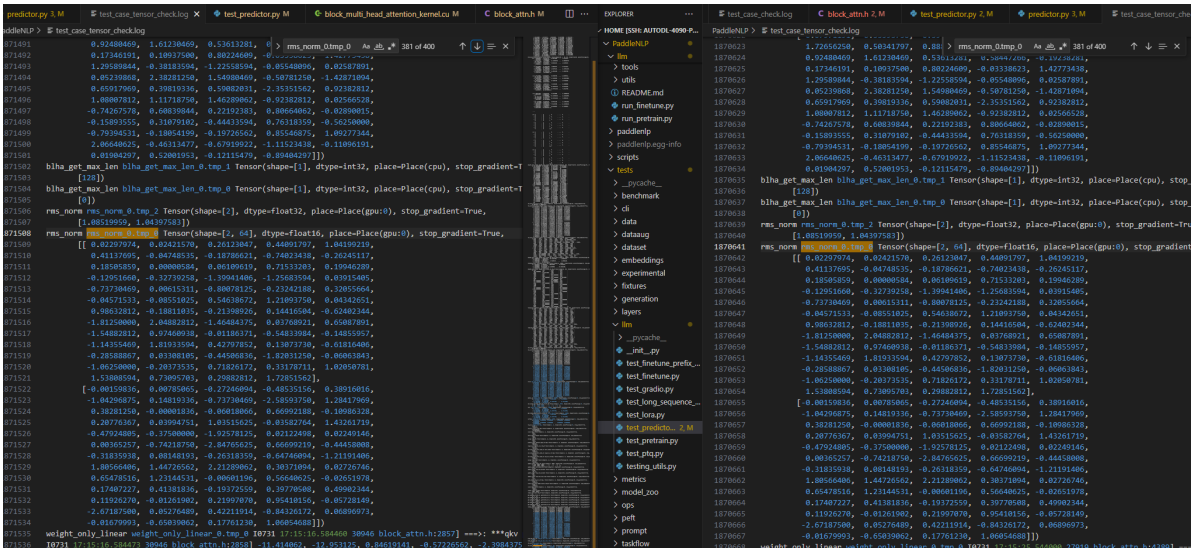


检查最后一个case（最后两个可以pass的input）引入误差的layer

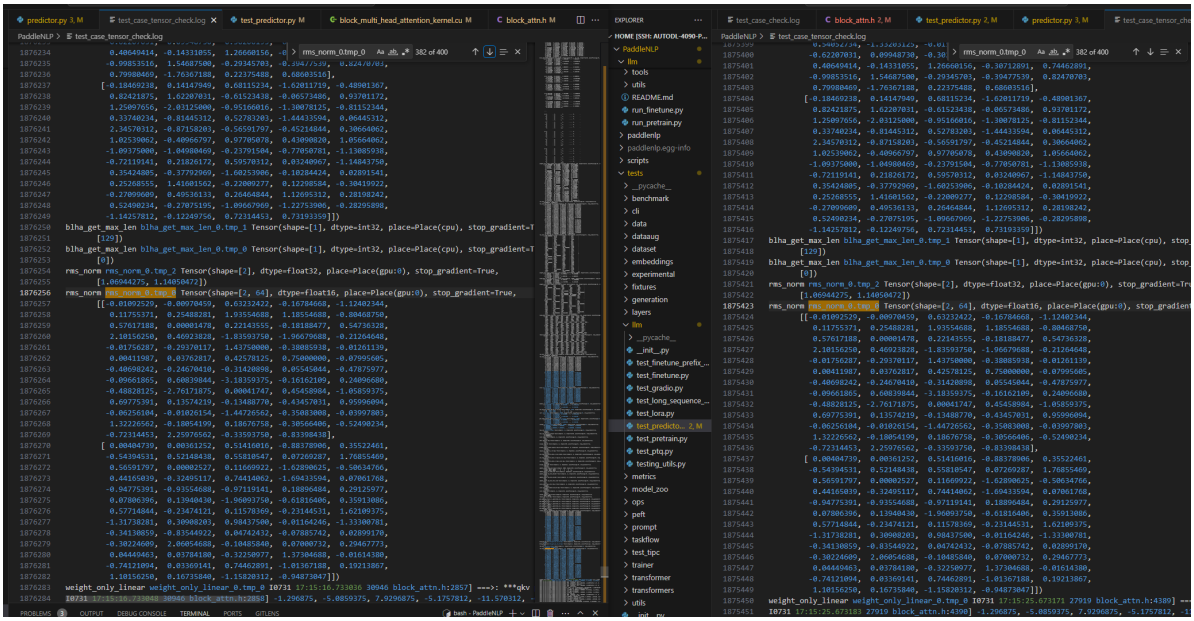
第一个matmul_v2_0.tmp_0（**有误差**）



第一个rms_norm_0.tmp_0（matmul_v2_0.tmp_0的输入）



第二个rms_norm_0.tmp_0（matmul_v2_0.tmp_0的输入）



打印test case

如果需要具体查看那个test_case pass或者fail了可以稍微修改一下 `test_wint8` 的代码如下

```
    def test_wint8(self):
        self.run_predictor({"inference_model": True, "quant_type":
"weight_only_int8", "block_attn": True})
        result_0 = self._read_result(os.path.join(self.output_dir,
"predict.json"))
        self.run_predictor({"inference_model": True, "quant_type":
"weight_only_int8"})
        result_1 = self._read_result(os.path.join(self.output_dir,
"predict.json"))

        assert len(result_0) == len(result_1)
        count, full_match = 0, 0
        i = 0
        for inference_item, no_inference_item in zip(result_0, result_1):
            min_length = min(len(inference_item), len(no_inference_item))
            count += int(inference_item[: min_length // 2] ==
no_inference_item[: min_length // 2])
            full_match += int(inference_item[:min_length] ==
no_inference_item[:min_length])
            i += 1
            if inference_item[:min_length] == no_inference_item[:min_length]:
                print("***NO %d PASS***\n"%i)
            else:
                print("***NO %d FAIL***\n"%i)

        print(" **len** :",len(result_0))
        print("***full_match: ***:", full_match)
        self.assertGreaterEqual(full_match / len(result_0), 0.75)

        if self.model_name_or_path == "__internal_testing__/tiny-fused-chatglm":
            self.assertGreaterEqual(count / len(result_0), 0.3)
        else:
            self.assertGreaterEqual(count / len(result_0), 0.4)
```